Towards Scalable 1024 Processor Shared Memory Systems

Robert B. Ciotti (ciotti@nas.nasa.gov),
NASA Advanced Supercomputing (NAS)
NASA Ames Research Center, Moffett Field, CA 94035

ABSTRACT: Over the past 3 years, NASA Ames has been involved in a cooperative effort with SGI to develop the largest single system image systems available. Currently a 1024 Origin3000 is under development, with first boot expected later in the summer of 2001. This paper discusses some early results with a 512p Origin3000 system and some arcane IRIX system calls that can dramatically improve scaling performance.

1 Introduction

Within the framework of a cooperative agreement between NASA and SGI, three (soon to be four) first-of-a-kind systems have been built and integrated into the NASA Advanced Supercomputing (www.nas.nasa.gov) Facility. Two of these machines run in the production supercomputing cluster[1,5] (a 256p and a 512p Origin2000). The third system, a 512-processor Origin3000, was brought up the second week of March 2001. Two of these 512p systems will be combined later in the summer of 2001 to form the first 1024p single system image supercomputer.

The primary goal of the project is to provide reliable platforms to run highly parallel applications that require tightly coupled processors. To that end, several elements are discussed that will significantly affect the outcome:

Interconnect/Topology - An interconnect topology that minimizes latency while providing ample bisection bandwidth.

Processing Elements - A processor that is competitive in sustained performance.

Programming Methodology - The development of a programming methodology that will allow efficient use of the machine.

Application Performance - Demonstration of sustained performance and scaling on a collection of important applications

System Availability - Reliability such that failures are infrequent enough so as not to interfere with day to day production use of the system.

2 Interconnect/Topology

The Origin3000 system is a cache-coherent non-uniform memory access (ccNUMA) computer. That is, all memory within a Single System Image (SSI) is globally addressable and cache coherent for all processors within that SSI. The effort at NAS will extend the coherency domain to as many processors as practical. Larger systems result in increasing latencies as the number of "hops" from one end of the system to another grows. In general, this latency is approximately 40ns/hop + 285ns. The best-case hop count on any Origin3000 system is 0 hops, or access to node local memory.

The NUMA design is quite flexible in that many different physical topologies can be constructed which are optimized for different design goals. As such, the 1024 system will be built in at least 2 different configurations. Consideration of additional factors beyond latency and bandwidth, such as how the system will function in a degraded mode, footprint, maintainability and cost will be weighed prior to moving forward with the final topology in late summer 2001.

At least three topologies have been proposed for the 1024 processor system:

Quad-Bristled Hypercube – This topology optimizes for cost by providing the minimum amount of interconnect hardware required. It can also be built today with no modification to the low level PROM code that "discovers" the physical layout of the machine at boot time. It suffers from a limited bisection bandwidth of 25 megabytes/second/processor and a higher worst-case latency of 12 hops. Its primary advantage is that it should work "out of the box".

QuadraTree[2] – This topology proposed by Ekechi Nwokah (nwokah@sgi.com) optimizes for latency by minimizing the radius of the system, a worst case of 6 hops, while providing a significant increase in bisection bandwidth over the Quad-Bristled Hypercube. It also provides for tightly integrated groups of 64 cpus for optimal OpenMP scaling.

Quad-Bristled Fat Tree[2] – This topology optimizes for bandwidth, while sacrificing some latency, 7 hops worst case with a bisection bandwidth double that of the QuadraTree. It provides for tightly integrated groups of 32 cpus for optimal OpenMP scaling.

Initially the system will be brought up in the Quad-Bristled Hypercube configuration in order to test basic functionality of the system and to work through any difficulties with the Operating System. As much as practical, tests will be conducted to determine which alternate topologies will provide the greatest benefit to applications. Any alternative topology will require that modifications be made to the PROM code.

3 Processing Elements

SpecFP numbers for single CPU performance indicate that SGI/MIPS processors are losing ground to other challengers in the market place (figure 1 – current as of May, 2001).

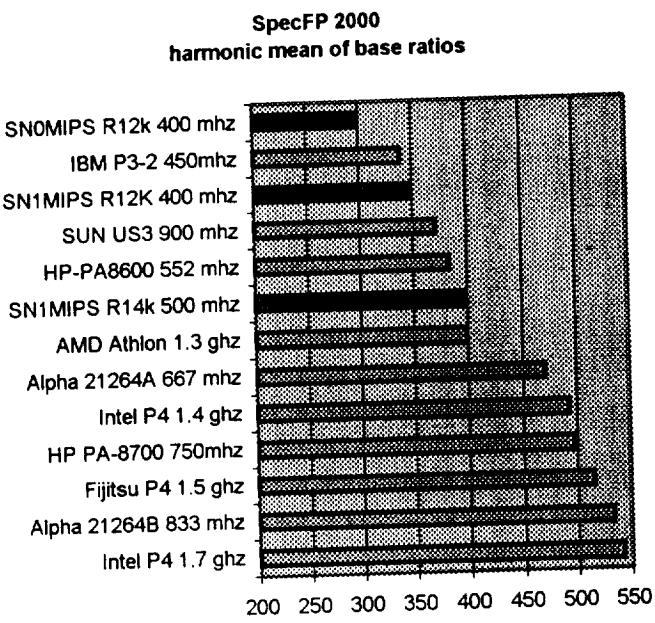

Figure 1

However, this view can be misleading in that the trend in high end computing is moving away from single processor clusters towards collections of multi-processor systems. Presumably because of the difficulty in scaling single processor message passing codes to hundreds or thousands of processors. The decision most computing centers face is not whether SMP's will be used, but how many processors the SMP will have. The performance picture changes significantly when looking at the SpecFP Throughput numbers (figure 2), which provides a more realistic representation of how large-scale system will perform under load, when all the processors are used simultaneously, memory references make it beyond the cache, and cooperating processes communicate with one another, One can conclude from this benchmark that higher sustained rates of performance are achieved via a more robust design of the memory system and that memory demand can be effectively managed up to 128 processors. Indeed, managing the memory traffic on 1024 way parallel applications will be critical in achieving high-sustained rates of performance. Based on previous results for Origin2000 systems, a performance target of 20% of peak (or at least 220 gigaflops) on a real world Computational Fluid Dynamics (CFD) design problem should be achievable once the system is upgraded from 400mhz to faster processors in early 2002.

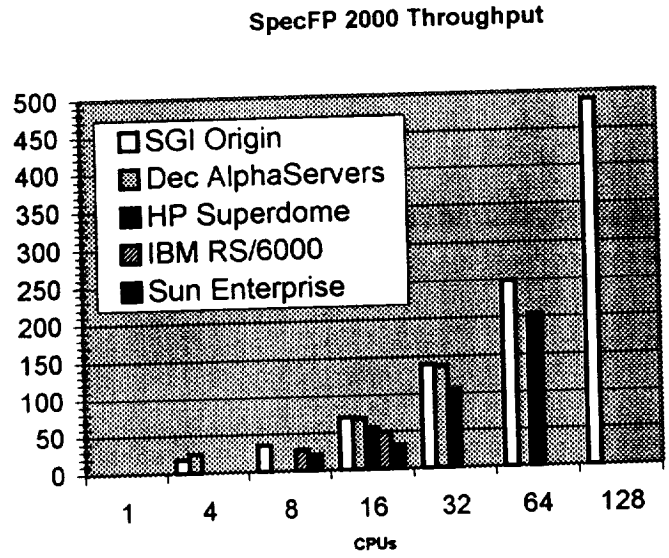

Figure 2

4 Programming Methodology

Shared memory has many benefits over explicit messaging. Some of those benefits include:

1. Decomposition of the problem is much easier, and more flexible than that required in an explicit messaging code.
2. The availability of a larger number of feasible parallelization strategies.
3. Large sections of code require no modification to port from earlier non-parallel or vector systems. For instance sections of code that deal with I/O may require little or no modification to work properly.
4. The average communication latency is less than other messaging approaches.
5. Fewer modifications are required to implement the parallelism.
6. Load balancing is much easier to manage and can be dynamic.

A disadvantage in the literature relates to the observed scaling of OpenMP. This is commonly (and incorrectly)

extrapolated to poor performance of shared memory systems in general. The success in achieving high levels of parallelism on shared memory systems (i.e., greater than 100 way parallel) has centered on the development of a technique conceived of by Jim Taft (jtaft@nas.nasa.gov) know as Multi Level Parallelism [3,6] (MLP). The MLP technique requires high-level coarse-grain decomposition similar to that for MPI[4], but communicates between these high level processes via shared memory instead of explicit messages. This minimizes the communication latency involved by eliminating any software protocol overhead as well as allowing any available hardware latency hiding mechanisms (e.g., cached writes, outstanding memory references, out of order execution, etc) to function. OpenMP[5] is then used to parallelize each of these high level decomposed pieces.

In many cases, however, successful application of the MLP technique requires the use of arcane, poorly documented, and undocumented system calls that must be made to manage thread placement and memory allocation, thankfully the MLP library hides this from the user. These calls and others were implemented in the MLP library by a local SGI site analyst Bron Nelson (bron@sgi.com), whose understanding and knowledge of IRIX made possible the success of this approach.

An early version of the MLP library contained a call to a routine called PID_TO_NODE written by John Richardson of SGI. The routine creates a memory locality domain (MLD) and associates it with the process id of the calling process. This has the effect of "advising" the kernel that pages allocated by this process are allocated on a specific node, and that this process is run on a CPU attached to that node. In MLP terminology, this is referred to as "pinning" or "PIN to Node". With the release of IRIX 6.5.10, some of the interfaces that these low level routines used were changed. Although the MLP library still worked, "pinning" did not. This resulted in temporarily losing performance on the Origin3000 systems.

5 Application Performance

At least three major applications have been ported to the MLP programming paradigm. INS3D[8,11] – an unsteady CFD code used for incompressible problems such as the design of the Space Shuttle main engine, the Data Assimilation System[9] (DAS), a major NASA climate research code based in part on CCM3, and OVERFLOW[7,10], a 3D compressible CFD code, used commonly throughout NASA and industry. All three applications show good scaling and performance well into the low to mid 100s of processors and all three depend upon MLP techniques to achieve this. Additionally, "pinning" (as discussed below) is required by all to achieve the best performance.

5.1 OVERFLOW - MLP

The OVERFLOW CFD code is available from NASA. It is maintained and distributed by Pieter Buening (p.g.buning@larc.nasa.gov) at NASA Langley. Subject to the NASA guidelines for technology export, one can contact Pieter and obtain a copy of OVERFLOW-MLP, be sure to specifically ask for the MLP version. The MLP version was developed by Jim Taft (jtaft@nas.nasa.gov) and can sustain over 60 gigaflops[10] on the 512 processor Origin2000. The MLP version maintains all the physics options of the standard release, but has the additional MLP functionality that includes both a runtime and dynamic load balancing capability.

Soon after the 512p Origin3000 system was brought on-line, Dennis Jespersen (jesperse@nas.nasa.gov) grabbed the latest and greatest MLP OVERFLOW source, and ran a 32.2 million grid point problem with 150 zones (figure 3). The figure shows the amount of time a single time step takes. As shown, the scaling on OVERFLOW basically stops at 64 processors.

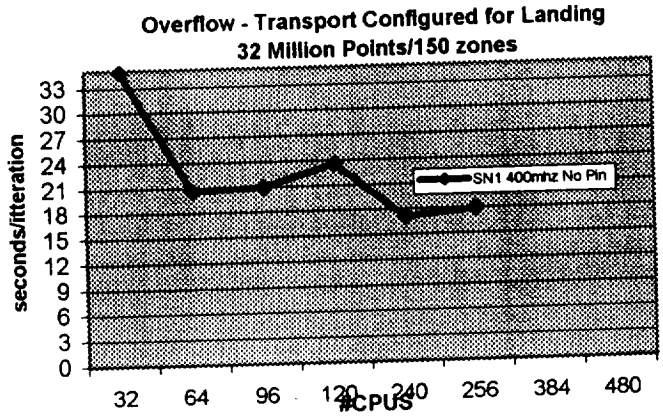

Figure 3

It was known that the pinning code did not work and Bron Nelson (bron@sgi.com) then set off to create a new and improved version of PID_TO_NODE for IRIX 6.5.10 and above called MP_ASSIGN_TO_CPU. The interface to this code is very simple and elegant, but what it does is crucial to scalability, and how it does it is not well known outside of SGI engineering.

5.2 Pin to Node via Mp_assign_to_cpu.c

Mp_assign_to_cpu.c is called is called from the forkit.f routine in the MLP library (the MLP library is available from Jim Taft of Sienna Software (jtaft@nas.nasa.gov)). The program is single threaded up to this point in the execution of the code. After the call to forkit, numpro MLP process will exist. In the call to this version of forkit.f, nthread is an array of integers, each of which defines the number of threads in each of the MLP

processes, numpro is an integer that represents the number of MLP processes to create, and nowpro returns to the newly created MLP process, its unique MLP id number:

```
      subroutine forkit3.f(nthread, numpro, nowpro)

c-----count offsets to starting cpus
      ival=0
      do n=1,numpro
        istart(n) = ival
        ival=ival+nthread(n)
      enddo

c-----spawn the threads - manual forks
      do n=2,numpro
      nowpid=getpid()
      if(nowpid.eq.master) then
                    ierror=fork()
                    endif
      nowpid=getpid()
      if(nowpid.ne.master) then
                    nowpro=n
                    go to 200
                    endif
      enddo

  200 call omp_set_num_threads(nthread(nowpro))

c-----pin to cpus

!$omp parallel
        call mp_assign_to_cpu3
                  (omp_get_thread_num(), istart(nowpro))
!$omp end parallel
...
```

As shown, each thread of each MLP process makes a call to mp_assign_to_cpu.c. As stated earlier, mp_assign_to_cpu.c makes some interesting systems calls roughly characterized by the following sequence of events:

1. Figure out which physical memory each cpu is attached to:

```
sysmp(MP_NUMA_GETCPUNODEMAP,
(void *)cpu_node_mapping, sizeof(cnodeid_t) * ncpus)
```

2. Figure out which physical processors the program has access to by looking at the CPUSET it is running. Also look at the NODEMASK to see which nodes are available. Map the NODEMASK nodes to CPUs and take the logical AND of the CPUSET and NODEMASK. Create an array from 1 to n of available cpus to assign threads to:

```
pmoctl(PMO_GETNODEMASK_UINT64,&sys_nodemask,
sizeof(sys_nodemask))
req.request = CPUSET_QUERY_CPUS;
sysmp(MP_CPUSET, &req)
```

3. Demand that physical pages be allocated from the memory attached to the CPU assigned to this thread:

```
mld_create(1,1024)
mldset_create(&mld, 1)
mldset_place(mldset, TOPOLOGY_PHYSNODES,
&affinity_info, 1, RQMODE_MANDATORY);
```

4. Lock the thread that called this routine to a cpu

```
sysmp(MP_MUSTRUN, cpulist[my_rank+starting_point])
```

The effect of the mp_assign_to_cpu call is shown in graphically in figure 4 if the forkit routine were called with the following: nthread=[8,13,...,16],numpro=n

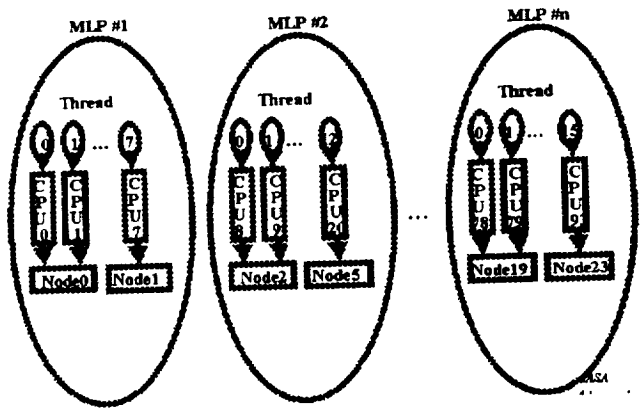

Figure 4

For example, OMP thread 1, of MLP process 2 should always run on CPU 9. Any memory allocated by this thread is placed on Node 2, the closest memory to CPU 9. This has the effect of improving locality. The technique is applicable to pure OpenMP codes, MPI codes, and multi-level MPI codes as well. Its effectiveness is dependent on the application and the number of CPUs in use, and it has demonstrated a dramatic on several codes worked on at NAS.

Figures 5 and 6 show the effect of the pin code vs the non-pin code (seconds per time step in figure 5 and speedup over 32 cpus in figure 6). As shown, scaling is significantly improved with gains all the way to 480 cpus. It is worthwhile to note that the load balancing for this problem was computed by the application as part of its initialization. This is quite significant and represents another big advantage of shared memory systems – load balancing is flexible and dynamic. With hand optimization of the load balancing, better speedup efficiencies can be achieved.

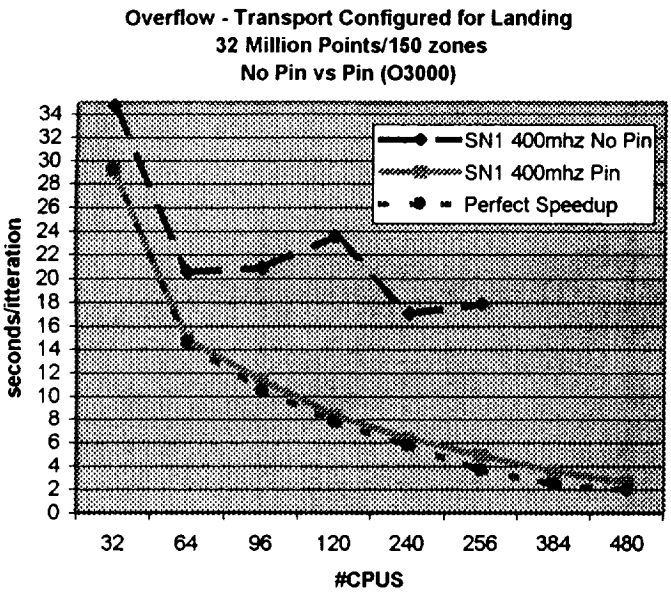

Figure 5

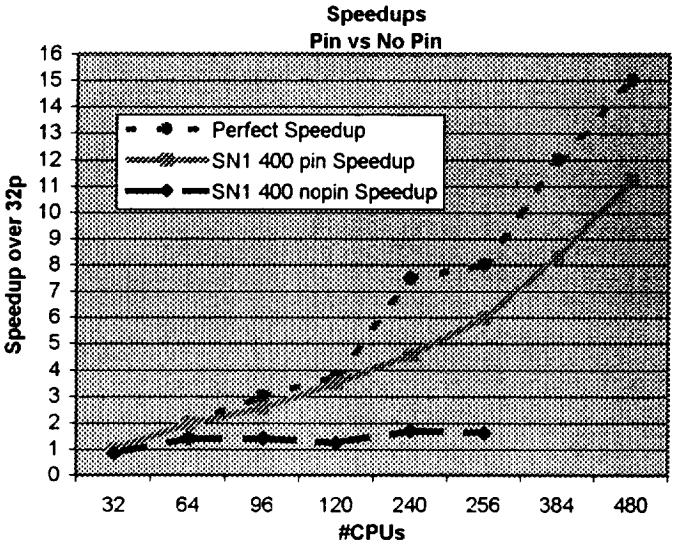

Figure 6

6 System Availability

In order for the 1024 processor system to be tolerable from a reliability standpoint it must run from 7 to 14 days between unscheduled interrupts, counting any system-related failure (i.e., these include hardware and software crashes, yet exclude outside causes such as a power failure). Based upon our experience with the other two systems[6] of this sort, reliability at this level is feasible. MTTI will be tracked over the life of the system and reported.

7 Conclusion

Progress continues to be made in making large scale shared memory systems work and in achieving high levels of performance on important NASA applications. To scale optimally beyond 64 cpus pinning techniques may be required.

8 References

1. J. Petersohn, K. Schilke, Experiences with the SGI/Cray Origin2000 256 Processor System Installed at the NAS Facility of the NASA Ames Research Center, 41st Cray User Group Conference Proceeding, May 24-28 1999.
2. E. Nwokah, QuadraTree topology for NASA Ames 1024p Origin3000 System, Internal communication, March 2001.
3. J. Taft, Multi-Level Parallelism, A Simple Highly Scalable Approach to Parallelism for CFD, HPCCP/CAS Workshop 98 Proceedings
4. W. Gropp, et al. Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press, Cambridge, MA, 1994.
5. OpenMP Architecture Review Board, OpenMP. A Proposed Standard API for Shared Memory Programming. October, 1997
6. B. Ciotti, J. Taft, J. Petersohn, Early Experiences with The 512 Processor Single System Image Origin2000, 42st Cray User Group Conference Proceedings, May 2000.
7. P.G.Buning, I.T. Chiu, S. Obayashi, Y.M. Rizk, and J.L. Steger, "Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent", AIAA-88-4359-CP, AIAA Atmospheric Flight Mechanics Conference, August 1988, Minneapolis, MN.
8. S. E. Rogers, D. Kwak, C. Kiris, "Numerical Solution of the Incompressible Navier-Stokes Equations for Steady-State and Time-Dependent Problems, AIAA Paper 89-0463, January 1989.
9. da Silva, S. J. Lin, The DAO Physical-space/Finite-volume Data Assimilation System Part I: Algorithm Theoretical Basis for the "Violet" Core System. DAO Office Note 2000-NN Version 1.0.3 Dated 11/22/1999
10. J taft, Achieving 60 Gigaflops with OVERFLOW-MLP, Parallel Computing. March 2001.
11. C. Kiris, D. Kwak, W. Chan., Parallel Unsteady Turbo-Pump Simulations For Liquid Rocket Engines. SC2000 conference proceedings, November 2000